Electronic Notes in Theoretical Computer Science 72 No. 4 (2003)
URL: http://www.elsevier.nl/locate/entcs/volume72.html 5 pages

Software Evolution through Transformations Preface and Workshop Summary

Reiko Heckel

Universität Paderborn, Germany
Email: reiko@upb.de

Tom Mens

Vrije Universiteit Brussel, Belgium
Email: tom.mens@vub.ac.be

Michel Wermelinger

Universidade Nova de Lisboa and ATX Software SA, Lisbon, Portugal
Email: mw@di.fct.unl.pt

1 Preface

Businesses, organisations and society at large are increasingly reliant on software at all levels. An intrinsic characteristic of software addressing a real-world application is the need to evolve. Such evolution is inevitable if the software is to remain satisfactory to its stakeholders.

Changes to software artifacts and related entities tend to be progressive and incremental, driven, for example, by feedback from users and other stakeholders. Changes may be needed for a variety of reasons, such as bug reports, requests for new features or, more generally, changes of functional requirements, or by the need to adapt to new technology, e.g., to interface to other systems. In general, evolutionary characteristics are inescapable when the problem to be solved or the application to be addressed belongs to the real world.

Transformations of artifacts like models, schemata, data, program code, or software architectures provide a uniform and systematic way to express and reason about the evolution of software systems. Literally, all activities that lead to the creation or modification of documents have a transformational aspect, i.e., they change a given structure into a new one according to pre-defined rules. In most cases, these rules manifest themselves as user-invoked operations in CASE tools or program editors. More abstract examples include rules for model refinement, for translating models to code, for recovering designs from legacy systems, or for refactoring and restructuring software.

1.1 Workshop Objectives

This workshop aimed at providing a forum for the discussion of transformational techniques in software evolution with particular focus on approaches that are generally applicable throughout the software development life-cycle. Thereby, we have distinguished two co-existing, complementary and mutually reinforcing views of the evolution theme. The more widespread view focuses on *the how of software evolution*, emphasising the methods and means by which software is evolved. The other focuses on *the what and why of the evolution phenomenon*, its nature and underlying drivers. Being mutually supportive, both views are required. A better understanding of the phenomenon leads to more appropriate ways of achieving evolution. Both views are supported through the general concept of *transformation*, i.e., the manual, interactive, or automatic manipulation of artifacts according to pre-defined rules, either as a conceptual abstraction of human software engineering activities, or as the implementation of mappings on and between modelling and programming languages.

1.2 Workshop Program

The workshop was scheduled for two half days and included one invited talk by *Stuart Kent* [1] as well as presentations of contributed papers in two regular sessions on different *Transformation Techniques* and on the mutual *Compatibility of Transformations*.

In addition, the workshop featured a special session on *Case Studies for Visual Modelling Techniques* held jointly with the Workshop on *Graph Transformation and Visual Modelling Techniques (GT-VMT 2002)* as part of the work carried out under the European research training network *SegraVis* (for *Syntactic and Semantic Integration of Visual Modelling Techniques*).

1.3 Acknowledgement

This workshop has been supported by the following projects: the scientific research network *Foundations of Software Evolution*, funded by the Fund for Scientific Research - Flanders (Belgium), the scientific research network *RELEASE* (for *REsearch Links to Explore and Advance Software Evolution*) funded by the European Science Foundation, and the European research training network *SegraVis*.

Reiko Heckel
Universität Paderborn, Germany

Tom Mens
Vrije Universiteit Brussel, Belgium

Michel Wermelinger
Universidade Nova de Lisboa and ATX Software SA, Lisbon, Portugal

2 Workshop Summary

The variety of problems addressed by the invited presentations and contributed papers underlines the generality of transformation as a concept to support and explain the evolution of software artifacts. Next, we summarise the issues raised by considering the following five questions.

2.1 What is transformed?

The artifacts subject to transformation include models, code, and data represented in different ways: concretely as XML documents [2], Smalltalk objects [3], or terms of a functional [2] or logic [4] programming language, or abstractly as graphs [6,5] or instances of a meta model [1].

Concrete representations are the basis for *implementing* transformations and thus providing tool support for (the *how* of) software evolution. Abstract representations are inevitable for reasoning in a uniform way about transformations which work at different levels and on different concrete representations, thus contributing to a general theory of the *what and why* of evolution.

In the discussion it has been observed that, even in a single approach, both views might be required, because a concrete internal representation should be provided with a more abstract, intuitive front end notation which makes it usable to a wider community.

2.2 How are transformations specified and applied?

In correlation with different representations of artifacts, different transformation techniques are applied and discussed: XSLT to transform XML documents [2,3], programmed transformations to manipulate Smalltalk objects [3], term rewriting [2], logic programming [4], and graph transformation [6,5].

Rule-based transformations are often non-deterministic. They may be applied in an automatic way, resulting in a translation or reduction process, or interactively like editor operations. Automatic transformations allow to free the human developer from tedious, mechanical tasks. Interactive transformations support the universal view of software development as manipulation of artifacts.

A general issue during the discussion of most presentations was the question, which set of features of a transformation language provides the right tradeoff between expressiveness on the one hand and the efficiency of execution or analysis on the other hand. Apparently, there is no general answer, but the question has to be decided for each individual application area.

2.3 Why are artifacts transformed?

Depending on *what* is transformed, the reason (i.e., *why*) for the transformation may be:

forward engineering: to instantiate a framework [4], transform a model into an implementation or refine it by a more detailed model [1], or to manipulate models or code, e.g., to support new features or correct errors [5];

reverse engineering: to recover the design of an existing system, e.g., by extracting a conceptual schema from a data base [2];

re-engineering: to improve models or code through refactoring without changing their semantics [6], and to transform the corresponding data to migrate it to the new version.

The reason for the transformation is closely related to the next question.

2.4 What is the semantics of transformations?

The semantics of transformations can be expressed as the relation between the semantics of the given and the transformed artifacts. It is therefore dependent on the semantics we associate with the entities we transform.

Refactorings, for example, are usually defined as behaviour-preserving transformation with respect to a notion of behaviour which is informally defined and which accounts only for certain aspects of the execution of a program, like access to variables, method calls, etc. Schema transformations may be classified according to the semantic relation between the given and the transformed schema into capacity preserving, extending, or reducing transformations, etc.

In general, the semantics of transformations, even if expressed informally, reveals the purpose of (otherwise purely structural) manipulations of models, program code, or data. Often, however, semantics is left implicit or, reversing the perspective, considered as a result of the transformation rather than a primary concept (for example, if transformations are used to describe refinement or equivalence of models).

2.5 How are transformations combined or coordinated?

Software development consists of transformations with different purpose and at different levels (see above). For example, re-engineering can be described as a combination of reverse engineering, design-level evolution, and forward engineering [2].

Co-evolution requires synchronised transformations of different views, or of models and code [6,5]. Refactorings may be applied at the level of code or of models, and in this case they have to be coordinated to preserve consistency.

It has been pointed out, in particular by the invited talk [1], that a major prerequisite for co-evolution is to keep track of the relations or mappings between the different artifacts that are transformed. In software engineering, this is known as traceability, like the possibility to justify the existence of a class by maintaining pointers to certain sections of a requirements document.

2.6 Summary of the Session on Case Studies

The session on case studies for visual modelling techniques was targeted at the three domain-specific objectives of the *SegraVis* research training network:

D1: modelling support for software evolution and refactoring;

D2: modelling of component-based software architectures;

D3: specification of applications with mobile soft- and hardware.

Beside a general discussion of the objectives, the session consisted in presentations of submitted case studies, two of which are contained in this volume. The first one [7], discussing possible evolution scenarios of a LAN simulation, was primarily aimed at objective D1, but allows an extension towards D3 based on a network with mobile nodes. During the discussion following the presentation, as a further case study it has been proposed to model and implement the set of refactoring rules given at `http://www.refactoring.com`, as they provide realistic and non-trivial examples of complex transformations which combine syntactic, semantic, and tool issues.

The second presentation was concerned with objectives D2 and D3, presenting a component-based model for a holonic manufacturing system with autonomous transport units [8]. In a third presentation, not present in these proceedings, Joost Kok introduced the *ArchiSurance* case, concerning the evolution of the IT system of a fictitious insurance company at the architectural level (thus touching objectives D1 and D2).

Generally it was observed that requirements for the structure and content of the case studies have to be stated more explicitly in order to reuse them, e.g., to assess the relative suitability of different approaches for the given problem.

References

[1] S. Kent. Model-Driven Language Engineering.

[2] J. Pérez, V. Anaya, J.M. Cubel, and J.Á. Ramos. Data reverse engineering of legacy databases to object oriented conceptual schemas.

[3] N. Revault. Model transformation based on production rules.

[4] T. Tourwé and T. Mens. High-level transformations to support framework-based software development.

[5] J. Padberg. Basic ideas for transformations of specification architectures.

[6] P. Bottoni, F. Parisi-Presicce, and G. Taentzer. Coordinated distributed diagram transformation for software evolution.

[7] D. Janssens, S. Demeyer, and T. Mens. Case Study: Simulation of a LAN.

[8] M. Klein, and M. Özhan, and M. Piirainen. Agent-Based Material Flow.